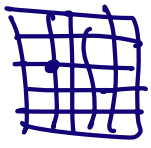
Welcome to Ma 221, Lec 10, Fall 24

Ex: Poisson's Eqn:

$$(*) \quad \frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} + q(x,y)\cdot u(x,y) = r(x,y)$$

in square, $x, y \in [0,1]$



discretize on 2D Mesh:

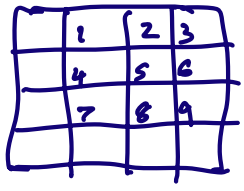
$$[x(i), y(j)] = [ih, jh] \quad h = \frac{1}{N+1}$$

simple: $q = r = 0$

approximate (*) by

$$\frac{1}{h^2}\Big[u(i-1,j) - 2u(i,j) + u(i+1,j) + u(i,j+1) - 2u(i,j) + u(i,j+1) \Big]$$

$$= \sum 4 \text{ neighbors} - 4u(i,j)$$



General Sparse Matrices:

Importance of ordering equations

what is best permutation P?

PLU, $P_r LUP_c$, $P_r LL^T P_r^T$

Cholesky stable for all P_r

Graph Theory

Def: A weighted, undirected graph G is a collection of 3 sets V, E, W

V = vertices (aka nodes)

E = edges connecting pairs of vertices (u, v), directed or undirected

undirected means (u,v) and (v,u) same
(u,u) allowed

W = weight (number) for each edge

Relationship to Sparse matrices

V = one row/col per vertex

E = locations of nonzeros
(u,v) means A(u,v) nonzero
undirected means $A = A^T$
(u, u) → A(u,u) nonzero

W = values of nonzeros

To build a matrix from graph
need to order vertices
lots of choices

Data Structures for Sparse Matrices

Goal: only store, operate on nonzeros

Simplest COO (Coordinate Format)

List of all nonzeros and locations

$$A = \begin{bmatrix} 2 & 0 & 7 & 0 & 5 \\ 0 & 1 & 4 & 0 & 3 \\ 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

$COO(A) = ((2,1,1),(7,1,3),(5,1,5), \cdots (8,3,3))$

Better: CSR (Compressed Sparse Row)

val: array of nonzeros in each row from row 1 to n, left to right

col_index: array of columns in which each nonzero in val lives

row_begin = pointer to start of each row

entries in row i live in row_begin(i) to row_begin(i+1)−1

val = (2, 7, 5, 1, 4, 3, 8)

col_index = (1, 3, 5, 2, 3, 5, 3)

row_begin = (1, 4, 7, 8)

also need m, n

roughly $\frac{2}{3}$ memory of COO

Analogous: CSC = compressed sparse column
like CSR on transpose

How do we pick best order of rows/columns
to minimize time, memory,
while being backward stable?

Start with Cholesky (any order stable)

Thm: picking best order to minimize
#flops (out of n! possibilities)
is NP-hard, exponential cost

⇒ need heuristics

RCM = Reverse Cuthill-McKee
= Breadth first search, reverse order

Def: A path in graph from v_1 to v_k
is a set of edges $(v_1, v_2), (v_2, v_3), \ldots$
(v_{k-1}, v_k)

Def: Distance between any 2 vertices
in Graph = length of shortest path
connecting them

RCM: ① pick any starting vertex, call it root

② compute distance from every other vertex to root

dist(root) = 0

dist(x) = 1 if (root, x) ∈ V

cost = O(nnz) = O(# nonzeros)

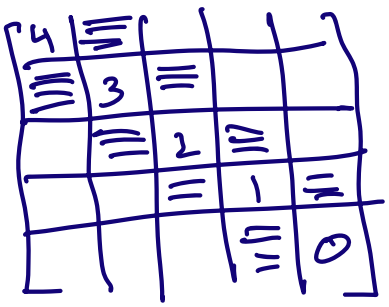
breadth first search

algorithm visits all vertices at distance 1 first, then distance 2, etc

③ order vertices in reverse of order visited

Fact: vertices at distance k can only be connected to vertices at distance $k-1$ or $k+1$, otherwise distance wrong

$\Rightarrow$ matrix block tridiagonal



max dist = 4

Matlab: symrcm

MD: minimum degree

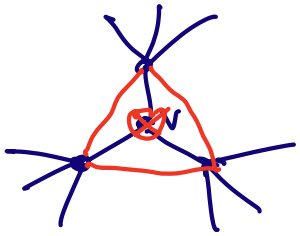
Def: degree of vertex = # edges it touches

Fact: If I pick vertex v as pivot, v has degree $d \Rightarrow d$ nonzeroes in its row and column

$\Rightarrow d^2/2$ multiplies and adds to take one step of Cholesky

$\Rightarrow$ fill in of at most $d^2/2$

Greedy Algorithm: at each step pick vertex of minimum degree (update degrees after each step)



deg(v) = 3

—— = filled in edge

neighbors become "clique"

Matlab: amd, symamd, colamd

ND: Nested Dissection:

Bisect vertices into 3 sets

$V = V_1 \cup V_2 \cup V_3$ (disjoint)

V_3 = separator

Goals: $|V_1| \sim |V_2|$

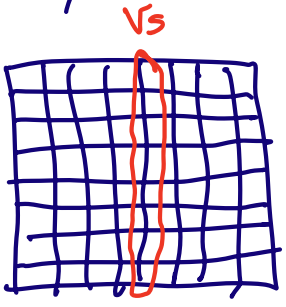
$|V_s|$ much smaller

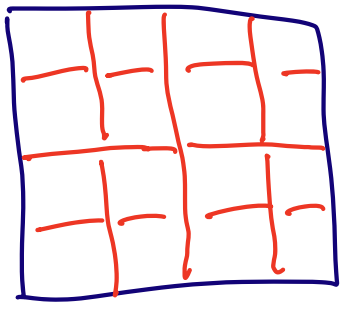
no edges from V_1 to V_2

Order vertices in V_1 first, then V_2, then V_s

$$A = \begin{array}{c} \\ V_1 \\ V_2 \\ V_s \end{array} \begin{array}{c} \begin{array}{ccc} V_1 & V_2 & V_s \end{array} \\ \begin{bmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{13}^T & A_{23}^T & A_{ss} \end{bmatrix} \end{array} \Rightarrow L = \begin{array}{c} \begin{array}{ccc} \sim\frac{n}{2} & \sim\frac{n}{2} & \text{small} \end{array} \\ \begin{bmatrix} L_{11} & 0 & 0 \\ 0 & L_{22} & 0 \\ L_{s1} & L_{s2} & L_{ss} \end{bmatrix} \end{array}$$

Apply bisection recursively to A_{11}, A_{22}





Thm (George, Hoffman, Martin, Rose; Gilbert, Tarjan, 70s-80s)

Any ordering for Cholesky on nxn mesh does $\Omega(n^3)$ flops

(from doing dense Cholesky on last V_s)

Attained by ND

Applies to planar graphs

(drawable on paper without edge crossings)

Thm (Ballard, D., Holtz, Schwartz, 2009)
words moved between main memory and cache for Cholesky is

$$\Omega\left(\frac{\#\text{flops}}{\sqrt{M}}\right) \quad M = \text{cache size}$$

For Cholesky on $n \times n$ mesh, $\Omega\left(\frac{n^3}{\sqrt{M}}\right)$

Thm (David, D., Grigori, Peyronnet, 2010)
Attainable by ND for mesh, done "carefully", bottleneck is last separator

Contrast with Bandsolver for $n \times n$ mesh

$$\#\text{flops} = O(bw^2 \cdot \text{dimension}) = O(n^4)$$

(more algs for ND in CS267)

What about 3D meshes? $n \times n \times n$

matrix has dimension n^3

dense Cholesky costs $O(n^9)$

band Cholesky costs $O(n^7)$

ND costs $O(n^6)$

Steps of Sparse Cholesky

Choose ordering (RCM, MD, ND, ...)
Build data structure for A and L
Perform factorization

Contrast with GE with pivoting:
partial pivoting could cause large fill in,
need other options.